A MULTIGRID PRECONDITIONER FOR THE SEMICONDUCTOR EQUATIONS[*]

JUAN C. MEZA AND RAY S. TUMINARO

Abstract. A multigrid preconditioned conjugate gradient algorithm is introduced into a semiconductor device modeling code, DANCIR. This code simulates a wide variety of semiconductor devices by numerically solving the drift-diffusion equations. The most time consuming aspect of the simulation is the solution of three linear systems within each iteration of the Gummel method. The original version of DANCIR uses a conjugate gradient iteration preconditioned by an incomplete Cholesky factorization. In this paper, we consider the replacement of the Cholesky preconditioner by a multigrid preconditioner. To adapt the multigrid method to the drift-diffusion equations, interpolation, projection, and coarse grid discretization operators need to be developed. These operators must take into account a number of physical aspects that are present in typical devices: wide scale variation in the partial differential equation (PDE) coefficients, small scale phenomena such as contact points, and an oxide layer. Additionally, suitable relaxation procedures must be designed that give good smoothing numbers in the presence of anisotropic behavior. The resulting method is compared with the Cholesky preconditioner on a variety of devices in terms of iterations, storage, and run time.

Key words. multigrid, semiconductors, drift-diffusion

AMS subject classifications. 35Q35, 65F10, 65F50

1. Introduction.

Currently most integrated circuits are designed in a 'trial and error' fashion. That is, prototypes are built and improved via experimentation and testing. In the near future it may be possible to significantly reduce the cost of building new devices by using computer simulations to shorten the design cycle. To accurately perform these complex simulations in three dimensions, however, new algorithms and high performance computers are necessary.

In this paper we discuss the use of multigrid preconditioning in conjunction with a conjugate gradient algorithm inside a semiconductor device modeling code, DANCIR [7]. DANCIR is a three-dimensional semiconductor device simulator capable of computing the solution of the steady-state drift-diffusion equations. The solution of the drift-diffusion equations involves the solution of a large nonlinear set of equations that arise from the spatial discretization of the drift-diffusion equations on a rectangular grid. These nonlinear equations are resolved using Gummel's method which requires three symmetric linear systems to be solved within each Gummel iteration. It is the solution of these linear systems that comprises the dominant computational cost of a simulation. The original version of DANCIR uses an incomplete Cholesky preconditioned conjugate gradient algorithm to solve these linear systems. Unfortunately, this algorithm has a number of disadvantages: 1) it can take many iterations to converge or it may not converge at all in some cases, 2) it can require a significant amount of computing time, and 3) it is not very parallelizable.

In this study we consider an alternate solution method based on a multigrid preconditioner. The multigrid method uses iterations on a hierarchy of grids to accelerate the convergence on the finest grid. The method requires interpolation, projection, and discretization operators for the different grids to be defined. Developing these operators in the context of the drift-diffusion equations requires some care due to the presence of greatly varying physical phenomena including wide scale variation in PDE coefficients and small scale phenomena such as contact points. In both cases, the development of operator dependent interpolation and projection is essential to improving the performance. Further, the presence of certain device characteristics such as oxide layers requires some care to insure that the different operators and the grid hierarchy adequately approximate the PDE on all levels. Finally, the presence of a severely stretched grid gives rise to anisotropic phenomena that requires a suitable relaxation procedure.

The paper is organized as follows. §2 describes the drift-diffusion equations that are most commonly used to model semiconductor devices. We also present the numerical methods currently used in a particular

[*] This work was supported by the Applied Mathematical Science Program, U.S. Department of Energy, Office of Energy Research and was performed at Sandia National Laboratories operated for the U.S. Department of Energy under contract No. DE-AC04-94AL85000.

simulation code developed at Sandia National Laboratories. In §3 we describe the multigrid preconditioner as well as some of the choices necessary to implement this algorithm for the drift-diffusion equations. In §4, the resulting method is compared with the incomplete Cholesky preconditioner on a variety of devices in terms of iterations, storage, and run time. We conclude in §5 with a summary of our results.

2. The Drift-Diffusion Equations.

The drift-diffusion model for semiconductor device modeling consists of a set of three, coupled, nonlinear partial differential equations: the potential equation plus two continuity equations, one each for the electron and hole current densities. For a complete derivation of the equations the reader can consult a variety of references, for example [9, 11, 13].

The potential or Poisson equation is given by

$$\epsilon \nabla \cdot E = -\epsilon \nabla^2 \psi = \rho, \tag{2.1}$$

where ϵ is the scalar permittivity of the semiconductor, ψ is the electric potential, and $E = -\nabla\psi$ is the electric field. The total electric charge density, ρ, is given by

$$\rho = q(p - n + N_D - N_A), \tag{2.2}$$

where q is the elementary charge, n is the density of free electrons, p is the density of holes, N_D is the density of donor impurities, and N_A is the density of acceptors.

The continuity equations for the electron and hole currents can be stated as

$$\frac{\partial n}{\partial t} - \nabla \cdot (\mu_n n E + D_n \nabla n) - R = 0, \tag{2.3}$$

$$\frac{\partial p}{\partial t} + \nabla \cdot (\mu_p p E - D_p \nabla p) - R = 0. \tag{2.4}$$

where μ_n and μ_p are the electron and hole mobilities respectively, D_n and D_p are the diffusion coefficients, and R is a term that accounts for the recombination and generation of electrons and holes. The mobilities and the recombination-generation term are functions of various physical and semi-empirical parameters and are usually modeled through a variety of sophisticated methods (see for example [11]).

2.1. Scaling of Variables.

The wide range in magnitude of both the dependent and independent variables creates difficulties in the numerical solution process. Independent variables such as the concentrations of impurity dopings N_D and N_A range from 10^{13} to 10^{19} carriers per cubic centimeter. Dependent variables such as the carrier densities n and p can range from 10^3 to 10^{19} carriers per cubic centimeter. The difficulties associated with the wide range in the magnitude of the dependent variables can be circumvented to a certain extent by employing different variables. However it has been noted by Polak [12] that changing variables amounts to trading high variability in the dependent variables for increased nonlinearity in the equations.

In the DANCIR code, the carrier concentrations are scaled by using the Slotboom variables, u and v:

$$n = n_{ie} \exp\left(\frac{q\psi}{kT}\right) u, \tag{2.5}$$

$$p = n_{ie} \exp\left(\frac{-q\psi}{kT}\right) v, \tag{2.6}$$

where n_{ie} is the effective intrinsic carrier concentration, k is the Boltzmann constant, and T is the bulk material temperature.

One of the advantages to using the Slotboom variables is that the current continuity equations assume the form of Poisson equations facilitating the numerical solution process. Using the Slotboom scaling the current densities can be written as

$$J_n = kT\mu_n n_{ie} \exp(q\psi/kT) \nabla u, \tag{2.7}$$

$$J_p = -kT\mu_p n_{ie} \exp(-q\psi/kT) \nabla v. \tag{2.8}$$

The resulting steady-state, drift-diffusion equations can then be expressed as

$$
\begin{aligned}
(2.9) \qquad f_1(\psi,u,v) &= \epsilon \nabla^2 \psi + q\left[n_{ie} e^{\frac{-q\psi}{kT}} v - n_{ie} e^{\frac{q\psi}{kT}} u + N_D - N_A\right] = 0, \\
(2.10) \qquad f_2(\psi,u,v) &= \nabla\cdot\left[kT\mu_n n_{ie} e^{\frac{q\psi}{kT}} \nabla u\right] + R = 0, \\
(2.11) \qquad f_3(\psi,u,v) &= \nabla\cdot\left[kT\mu_p n_{ie} e^{\frac{-q\psi}{kT}} \nabla v\right] + R = 0.
\end{aligned}
$$

2.2. Nonlinear Equations. If the functions f_1, f_2, and f_3 are defined by equations (2.9–2.11) then the nonlinear system of equations arising from the spatial discretization of the drift-diffusion equations can be written more compactly as: $F(\psi,u,v) = [f_1, f_2, f_3]^T = 0$. The DANCIR code uses Gummel's method [6] (also known as nonlinear block Gauss-Seidel) to solve this nonlinear equation. One Gummel iteration consists of solving f_2 for u, f_3 for v, and then f_1 for ψ. Gummel's method has the advantage of only having to solve three linear systems at each iteration. The disadvantage is that convergence can be quite slow in certain circumstances, for example in high voltage situations.

In practice, the solution of the steady-state, drift-diffusion equations is accomplished by solving a series of continuation steps where each continuation step is in turn a steady-state problem. The initial steady-state or equilibrium problem solved is that of the device with no external voltages applied. The potential at the contacts is then incremented until the desired voltage is reached at the contacts. The initial estimate for the potential is computed by solving a nonlinear potential equation. The DANCIR code uses a Newton method for this calculation because the Jacobian is symmetric in this special case thereby not incurring any extra expense for storage over the Gummel iteration.

2.3. Linear Equations. Within each Gummel iteration, 3 linear systems of equations must be solved. As we mentioned above, the use of the Slotboom variables transforms the continuity equations for the electron and hole currents into a set of self-adjoint partial-differential equations. The linear systems resulting from the discretization are therefore symmetric and positive definite. The particular method used in the original version of the DANCIR code is a preconditioned conjugate gradient method with an incomplete Cholesky factorization used as the preconditioner [4]. The solution of these linear systems usually constitute the dominant amount of work so any improvement in this part of the code would have a significant effect in overall performance.

In the remainder of this paper, we discuss the replacement of this preconditioner by a multigrid preconditioner and make comparisons between the two preconditioners.

3. Multigrid Preconditioner. The multigrid algorithm is a fast and efficient method for solving the systems of equations that arise from many PDE applications. We give only a brief sketch of one type of multigrid algorithm. Detailed descriptions of more general multigrid algorithms can be found in [3, 5].

One iteration of a simple multigrid 'V' cycle consists of smoothing the error using a relaxation technique (such as Gauss-Seidel), 'solving' an approximation to the smooth error equation on a coarse grid, interpolating the error correction to the fine grid, and finally adding the error correction into the approximation (and perhaps performing some additional relaxation steps). An important aspect of the multigrid method is that the coarse grid solution can be approximated by recursively using the multigrid idea. That is, on the coarse grid, relaxation is performed to reduce high frequency errors followed by the projection of a correction equation on yet a coarser grid, and so on. Thus, the MG method corresponds to a series of relaxation iterations on a hierarchy of grids with different mesh sizes followed by the use of a direct solver on the coarsest grid (which is usually a fairly small grid). We summarize one iteration of this procedure in Figure 1.

It is important to note that when multigrid is used as a preconditioner within the conjugate gradient method, it is necessary that the preconditioner be symmetric. This can be accomplished in the above procedure by choosing the postrelaxation ('relax2' in Figure 1) as the transpose of the prerelaxation ('relax'

```
Procedure MG($b, u, level$)
    if ($level$ == COARSEST ) then $u \leftarrow A_{level}^{-1} b$
    else
        $u \leftarrow$relax($b, u, level$)
        $r \leftarrow b - Au$
        $\tilde{r} \leftarrow Rr$          /* R is a projection operator */
        $v \leftarrow 0$
        $v \leftarrow$ MG($\tilde{r}, v, level + 1$)
        $u \leftarrow u + Pv$    /* P is an interpolation operator */
        $u \leftarrow$relax2($b, u, level$)
    endif
```

FIG. 1. *MG Algorithm for $A_{level} u = b$*

in Figure 1) [8]. The Jacobi iteration is one smoother which has this property when it is used for prerelaxation and postrelaxation. Another smoother combination with this property consists of using red-black Gauss-Seidel for prerelaxation and black-red Gauss-Seidel for postrelaxation.

To complete the definition of the above method, we will define a grid hierarchy, grid transfer operators, a coarse grid discretization scheme, and a specific relaxation method.

3.1. Grid Hierarchy. In the DANCIR code, the discretization mesh used consists of a tensor product grid of one-dimensional arrays. In defining a grid hierarchy for the multigrid method, we consider only coarsening by a factor of 2 to facilitate code development. It should be noted that in order to carry out this procedure it is necessary to restrict the size of the fine grid. Specifically, for two-dimensional simulations, the fine grid must contain $(2^k m + 1) \times (2^j n + 1)$ points where $max(k, j)$ defines the number of levels in the hierarchy and $(m + 1) \times (n + 1)$ is the size of the coarsest grid. For the case $k \geq j$, the grid hierarchy is defined as follows:

$$\begin{array}{lcl}
\mathcal{G}_0 & : & 2^k m + 1 \ \times \ 2^j n + 1, \\
\mathcal{G}_1 & : & 2^{(k-1)} m + 1 \ \times \ 2^{(j-1)} n + 1, \\
 & & \vdots \\
\mathcal{G}_j & : & 2^{(k-j)} m + 1 \ \times \ n + 1, \\
 & & \vdots \\
\mathcal{G}_k & : & m + 1 \ \times \ n + 1.
\end{array}$$

Though this scheme is straight-forward, some care must be taken due to certain device characteristics such as the presence of an oxide layer as well as the contacts. This difficulty will be discussed after defining the grid transfer operators and the coarse grid discretization.

3.2. Grid Transfer Operators. In simple multigrid codes it is quite common to use linear interpolation for second order problems and full-weighting or half-weighting for projection [3]. In most cases these simple grid transfer operators are sufficient and the resulting multigrid scheme works quite well. However, when the PDE coefficients vary greatly (or contain a discontinuity), these choices for the grid transfer operators may not be sufficient. To illustrate this point consider the simple PDE

$$(w(x)\ u_x\ )_x \ = f(x), \qquad 0 < x < 1, \tag{3.12}$$

with Dirichlet boundary conditions at $x = 0$ and $x = 1$ and

$$w(x) = \begin{cases} \varepsilon & x < .5 \\ 1 & x \geq .5 \end{cases} .$$

If linear interpolation (or even higher order interpolation) is used to obtained the value of u at $x = .5$, the resulting quantity $w(x)u_x$ will in general be discontinuous. Since this term is differentiated, it is quite clear that this discontinuity is undesirable. That is, interpolating such that u is smooth (i.e. u_x is constant) at the interpolation point is not the right criterion. Instead, we need to take into account the function $w(x)$. One possibility (see [14] or [1]) is to require that the term $w(x)u_x$ be constant at the interpolated points. This results in an interpolation formula of the form

$$u(x) = \frac{w(x-\frac{h}{2})u(x-h) + w(x+\frac{h}{2})u(x+h)}{w(x+\frac{h}{2}) + w(x-\frac{h}{2})}, \tag{3.13}$$

for the above example on a uniform grid (with mesh spacing h on the fine grid). This formula can be verified by observing that when it is used, the central difference approximations to $w(x)u_x$ at $x - \frac{h}{2}$ and $x + \frac{h}{2}$ are equal. It should be noted that the above interpolation operator could have been defined on simply algebraic terms. Specifically, when (3.12) is discretized in a standard way the resulting difference operator is tridiagonal. To solve this tridiagonal system, a procedure called cyclic reduction can be used [4]. This procedure is essentially Gaussian elimination with a special elimination ordering of the points. The first step corresponds to eliminating all the even numbered points (where the points are numbered sequentially along the line). If we make an analogy between the multigrid algorithm and cyclic reduction, the interpolation algorithm corresponds to one step in the back solve of cyclic reduction.

Generalization of the interpolation procedure given above to two or three-dimensional PDE's is not obvious as the exact analogy with cyclic reduction is no longer possible. In this work we consider the following interpolation procedure in 2D.

1. Split the finite difference operator on $\mathcal{G}_l$, A_l, into matrices corresponding to the x, and y derivatives as well as the 0th order terms:

$$A_l = A_l^x + A_l^y + G_l.$$

2. Average the derivative operators in the direction orthogonal to the derivative:

$$\tilde{A}_l^x(j) = \frac{1}{4}A_l^x(j-1) + \frac{1}{2}A_l^x(j) + \frac{1}{4}A_l^x(j+1),$$

   where $A_l^x(j)$ is the x derivative operator for level l on horizontal line j of the grid.
3. Interpolate in the x direction. For points where the coarse and fine grid coincide, use injection. For points in between 2 coarse grid points use the one-dimensional procedure outlined above in conjunction with $\tilde{A}_l^x(i)$. Specifically, set the interpolated value at (i, j) such that

$$\{\tilde{A}_l^x(j)u\}_i = 0,$$

   where u is the vector of interpolated values corresponding to line j of the grid. This essentially corresponds to the back solve step of cyclic reduction applied to $\tilde{A}_l^x(j)$. It is important to note that at this point the operator u is defined on every other horizontal line.
4. Repeat the procedure in the y direction for the horizontal lines that have not yet been defined in the previous x interpolation. In particular, for fine grid points which are surrounded by 4 coarse grid points, use the interpolated values from the x interpolation.

Effectively, this procedure corresponds to performing the interpolation in one direction after another using one-dimensional difference operators defined from the original difference operator. The overall procedure is similar to that in [1] where they use an arithmetic average of the harmonic averages (as opposed to the harmonic average of the arithmetic averages) to define the interpolation operator.

For the projection operator we simply use the transpose of the interpolation scheme. Given that the difference operator is symmetric, this is a quite natural way to define a weighted projection operator. Using this projection on a one-dimensional problem corresponds to performing the forward elimination step of a cyclic reduction procedure on the tridiagonal discretization matrix.

3.3. Coarse Grid Discretizations. In typical multigrid codes, there are two ways of obtaining PDE discretizations for the coarser grids. One reliable technique, Galerkin coarsening, uses the interpolation and projection operators.

While this procedure works well for two and three-dimensional problems, it has several disadvantages. For example, when standard interpolation and projection operartors are used the Galerkin procedure may result in a larger difference stencil on coarser grids. Thus the programmer would have to write new codes for the coarse grid discretization as well as incur an extra storage penalty.

An alternative to the Galerkin procedure is to use an averaging procedure to generate PDE coefficients in conjunction with the differencing scheme used on the fine grid. There are many possible averaging procedures. In this work, we use a scheme based on the 1D Galerkin operator. Specifically, when a Galerkin procedure is used in conjunction with the operator dependent interpolation and projection on (3.12), the resulting coarse grid operator is equivalent to the reduced operator obtained with 1 step of cyclic reduction. For (3.12), this coarse grid operator is equivalent to a three point difference operator which corresponds to using the fine grid difference scheme on the coarse grid in conjunction with the PDE coefficients

$$\tilde{w}(x) = \frac{2\ w(x+h)\ w(x-h)}{w(x+h) + w(x-h)}.$$

For higher dimensional problems we first split the finite difference operator into different terms. A standard averaging procedure is used to approximate the constant term, G_l, on the coarse grid. For the derivative terms, we first form averages (e.g. $\tilde{A}_l^x(i)$) as for the interpolation. Then, we use a cyclic reduction procedure on the 1D problems in each of the different coordinate directions to generate approximations to the PDE coefficients on the coarser grid. Overall, the resulting procedure can be viewed as an approximation to the reduced equations of cyclic reduction (or the Galerkin coarsening) where the operator has first been split into component terms, the cyclic reduction procedure has been applied, and then the terms are regrouped to form the coarse grid discretization.

3.4. Relaxation Method. There are two major possibilities for the relaxation method: point relaxation and line relaxation. It is well-known in the multigrid community that line relaxation (or semicoarsening) should be adopted when the method is applied to anisotropic problems. That is, while standard point relaxation methods sufficiently smooth the error for the Poisson equation, their performance is quite poor for severely anisotropic problems. If, however, line relaxation is used instead of the point relaxation, it is once again possible to obtain good multigrid convergence rates even for anisotropic problems. We omit the details and refer the reader to [3] where a Fourier analysis is given illustrating this phenomenon. In our case, the highly stretched grid (e.g. Figure 2) is a source of anisotropic phenomena.[1] Thus, we have implemented red/black Gauss-Seidel as well as red/black alternating line Gauss-Seidel for prerelaxation and black/red point and line Gauss-Seidel for postrelaxaton. Additionally, we have implemented local Gauss-Seidel procedures to take into account the physics of the simulation. In particular, the majority of the difficulties for the smoother are caused by the top of the device (near the contacts, oxide layer, and doped regions). The local Gauss-Seidel procedures consist of performing red/black (point or alternating line) Gauss-Seidel on the residual equation restricted to the domain covering the upper 1/4 of the device. This local procedure is usually used after the standard relaxation procedure to improve the smoother in the upper part of the device (at only 1/4 the cost of the global procedure).

[1] It is not clear whether line relaxation would be necessary with another grid structure. That is, if the sole source of the anisotropic behavior is the grid, it may not be necessary to use line relaxation.

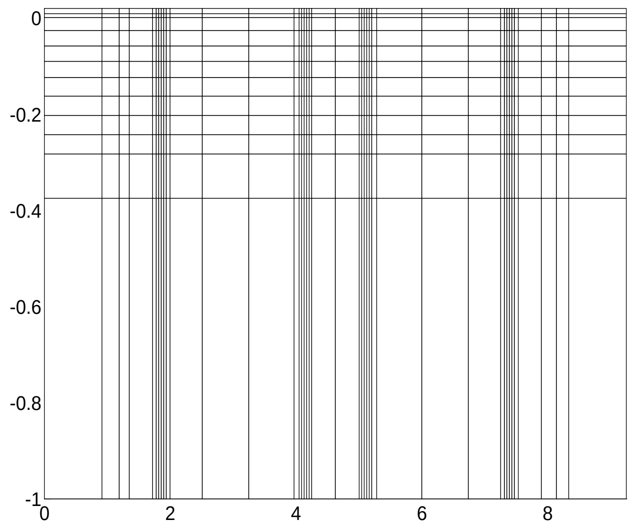


FIG. 2. *Grid obtained by keeping every eighth line from a typical MOSFET grid.*

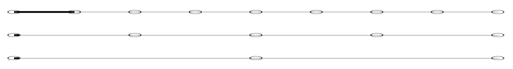


FIG. 3. *1D grid hierarchy*

3.5. Contact Points and Oxide Layer. In defining the grid hierarchy no particular attention has been given to device characteristics such as the oxide layer and the contact points present in many devices. It is entirely possible that the location of the boundary between the oxide and the interior or the boundary between a contact point and the interior could differ between the fine and coarse grids. For example, consider the one-dimensional hierarchy of 3 grids shown in Figure 3. If we have a contact region (denoted in bold) which covers 2 points on the fine grid, then it effectively becomes smaller on the next coarser grid. To prevent this phenomenon, there are two possible solutions. One option is to choose the grid hierarchy in conjunction with the contacts (and oxide) such that the boundaries of these regions remain in place. The other possibility, which we consider, is to keep the grid hierarchy but to modify the discretization and right hand side on the coarse grid such that the location of the contact boundary is maintained. For example, the matrix equation corresponding to the fine grid above

$$\begin{pmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & -2 & 1 & & & & & \\ & & 1 & -2 & 1 & & & & \\ & & & 1 & -2 & 1 & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 1 & -2 & \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{pmatrix}$$

has the same solution at the odd points as

$$\begin{pmatrix} 1 & & & & \\ & \mathbf{-3} & \mathbf{1} & & \\ & 1 & -2 & 1 & \\ & & 1 & -2 & \\ & & & & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_3 \\ u_5 \\ u_7 \\ u_9 \end{pmatrix} = \begin{pmatrix} b_1 \\ \mathbf{2b_3 + b_4} \\ b_4 + 2b_5 + b_6 \\ b_6 + 2b_7 + b_8 \\ b_9 \end{pmatrix}.$$

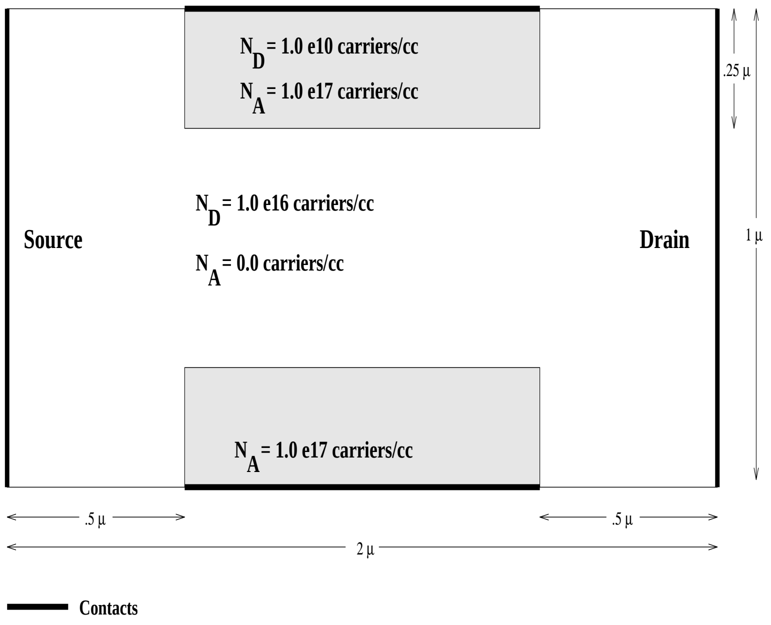
Notice that the main change is the second equation (discretization and right hand side projection). This procedure corresponds to cyclic reduction and can also be motivated by trunction error arguments. The main point is that the operator dependent schemes automatically take care of these situations to maintain a proper coarse grid approximation. Of course, this example is one-dimensional and in higher dimensions it will not be possible to have an exact representation on the coarse grid. However, by using the operator dependent grid transfers and discretizations we need not worry too much about the grid hierarchy with respect to the oxide and the contacts. Further, any small degradations in the coarse discretization (for example near the corners of an oxide region) will in general not cause great difficulties for a conjugate gradient routine (as the problem areas will be of very low rank).

4. Algorithm Summary. To summarize, the numerical algorithm used for the nonlinear equations is the Gummel iteration. Gummel's method requires 3 linear system solutions. For these a conjugate gradient algorithm is used in conjunction with a multigrid preconditioner. Finally, within the multigrid preconditioner, Gauss-Seidel is used as a smoother with red-black sweeps for prerelaxation and black-red for post relaxation. It should be noted that it is possible to use multigrid as a solver instead of a preconditioner (i.e. without the conjugate gradient iterations). However, in our experiments we found that the multigrid preconditioned conjugate gradient code out performed the use of just multigrid. We believe that there are two primary reasons for this:

- The presence of small-scale effects such as small contacts and doping regions which disappear on the coarse grid degrade the multigrid coarse grid operator.
- The interpolation, projection and coarse grid scheme used are a bit simplistic. These choices were made partially because they were easy to incorporate into a large already existing program with fairly complicated data structures and a simplistic scheme for handling nonrectangular grids (resulting from the oxide region). Unfortunately these procedures do not very well handle the corners of the oxide.

While these low rank degradations can make the multigrid convergence slower, they do not greatly affect the conjugate gradient procedure.

5. Numerical Results. A two-dimensional version of the multigrid algorithm described in the previous section was incorporated into the DANCIR code. To evaluate the multigrid scheme, a number of experiments have been performed on a variety of devices. Before illustrating these results we briefly describe the devices used in the comparison.



N_D = 1.0 e10 carriers/cc
N_A = 1.0 e17 carriers/cc
N_D = 1.0 e16 carriers/cc
N_A = 0.0 carriers/cc
Source
Drain
N_A = 1.0 e17 carriers/cc
.25 µ
1 µ
.5 µ
.5 µ
2 µ
Contacts

FIG. 4. *JFET Device*

5.1. JFET. The first model used consists of a junction field effect transistor (JFET) depicted in Figure 4. The simulation is started with zero volts on all of the contacts. Once the equilibrium solution is computed, the voltage on the bottom and top contacts is gradually incremented (and the corresponding steady state solutions are computed) until the final contact voltage is reached (1.0 Volts). After this stage the drain contact voltage is incremented until it attains a value of 1.0 Volts. The current-voltage characteristics (computed with the DANCIR/MG code) are displayed in Figure 5.

5.2. MOSFET1. The second test problem consists of a simple MOSFET device depicted in Figure 6. In this case the simulation starts at equilibrium. The gate contact is first incremented to 5.5 Volts and finally the drain value is incremented to 2.5 Volts.

The current-voltage characteristics (computed with the DANCIR/MG code) are displayed in Figure 7.

5.3. MOSFET2. This final device is our most complex device and corresponds to a 1.25μ N-channel MOSFET which has been used at Sandia for testing purposes. This device is depicted in Figure 8. In this case the simulation starts at equilibrium. The gate contact is first incremented to 5.5 Volts and finally the drain value is incremented to 2.5 Volts.

The current-voltage characteristics (computed with the DANCIR/MG code) are displayed in Figure 9. These characteristics are standard for this device.

In Tables 1 - 3 we illustrate the total number of iterations for the conjugate gradient method and the cpu time corresponding to an entire simulation where the conjugate gradient iterations for each linear solve terminate when the residual is reduced by 10^{-9}. In parenthesis we indicate the average number of conjugate gradient iterations per linear solve. It should be noted that a less strict convergence criterion for the linear subiterations is not considered in this paper. While a milder criteria may be more efficient for the overall nonlinear problem[2], our focus is to compare linear solvers where it is best if both simulations

[2] In our experience, we have found that it can often be more efficient to perform just a few multigrid sweeps for the two carrier equations. However, it is necessary to solve somewhat accurately the Poisson equation for the potential. This is due to the fact that the potential is exponentiated in the Gummel iteration. Thus, if the potential is not accurate (and these inaccuracies are amplified due to exponentiation), the nonlinear path taken by the Gummel iteration can be suboptimal.

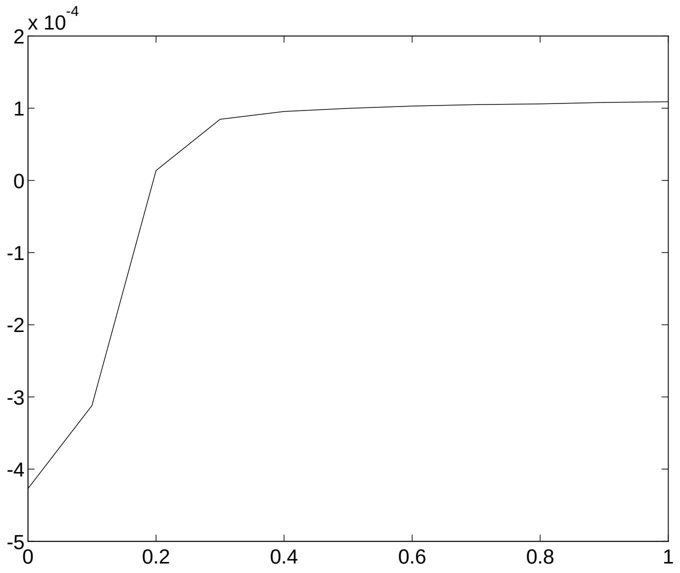


FIG. 5. *Current-voltage characteristics for JFET.*

Preconditioner	49×33				321×97			
	Poisson		Continuity		Poisson		Continuity	
	its	time	its	time	its	time	its	time
ILU	1499 (13)	.4	11351 (48)	2.9	5243 (42)	25.0	49635 (199)	231.1
MG(1L,0L,4)	726 (6)	0.7	1959 (8)	1.9	1270 (10)	29.2	2182 (9)	50.4
MG(1L,1L,4)	562 (5)	0.7	1750 (7)	2.3	1008 (8)	30.3	1750 (7)	52.9
MG(2L,2L,4)	437 (4)	1.0	1499 (6)	3.3	840 (7)	41.0	1502 (6)	73.4
MG(1L,3L,4)	456 (4)	0.9	1589 (7)	3.0	815 (7)	36.0	1488 (6)	65.9
MG(3P,0P,4)	963 (8)	0.8	4133 (18)	3.3	1192 (10)	23.5	1827 (7)	36.5
MG(5P,5P,4)	626 (5)	1.0	2650 (11)	4.1	973 (8)	34.6	1555 (6)	55.9
MG(1L,2L,4)	503 (4)	0.8	1682 (7)	2.7	887 (7)	33.0	1543 (6)	57.6
MG(1L,2L,5)	503 (4)	0.8	1682 (7)	2.7	887 (7)	31.5	1550 (6)	55.1
MG(1L,2L,6)	503 (4)	0.8	1682 (7)	2.7	887 (7)	31.3	1606 (6)	56.7
MG(1L,0L,5)	726 (6)	0.6	1959 (8)	1.7	1273 (10)	23.7	2191 (9)	40.9
MG(1L,0L,6)	726 (6)	0.6	1959 (8)	1.7	1273 (10)	23.7	2214 (9)	41.3

TABLE 1
JFET results: iterations and time (in minutes) for 23 Gummel iterations.

(multigrid and incomplete Cholesky preconditioned) follow the same nonlinear path. In the tables the notation $MG(\#1X, \#2Y, \#3)$ indicates that the multigrid preconditioner uses #1 pre- and post-relaxation iterations[3] of the 'X' (P for point or L for alternating line) Gauss-Seidel procedure and #2 pre- and post-relaxation iterations of the 'Y' (P for point or L for alternating line) local Gauss-Seidel procedure on each grid in the #3 level hierarchy. All of the ILU results corresponds to $ILU(0)$ which has the same sparsity pattern

[3] Actually, the post relaxation operator is the transpose of the prerelaxation operator. Thus, the prerelaxation operator uses red/black Gauss-Seidel while the postrelaxation operator uses black/red Gauss-Seidel.

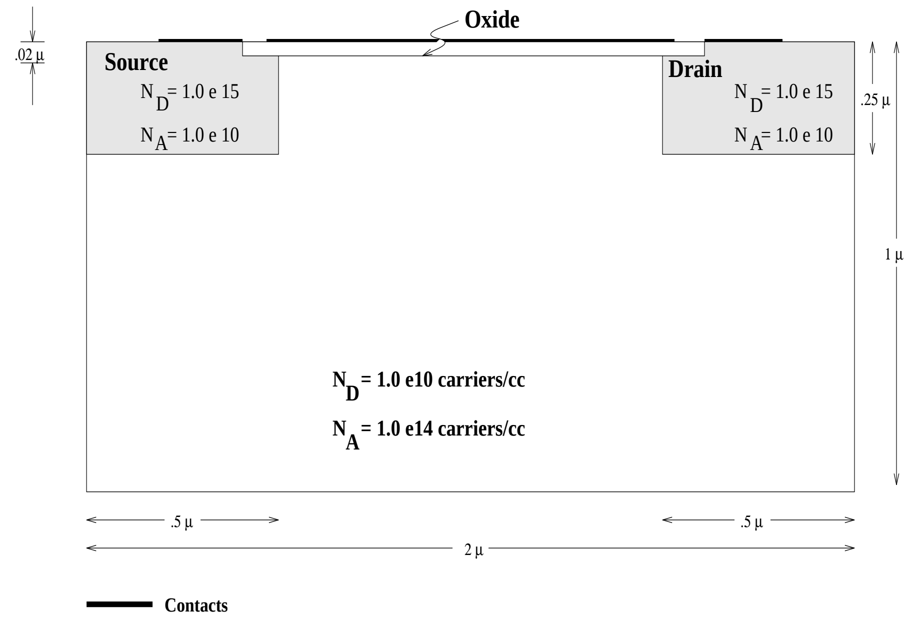


FIG. 6. *MOSFET1 Device*

Preconditioner	129 × 129			
	Poisson		Continuity	
	its	time	its	time
ILU	21703 (139)	53.9	61504 (197)	152.6
MG(1L,0L,4)	1635 (10)	17.4	3755 (12)	39.7
MG(1L,1L,4)	1523 (10)	20.8	3369 (11)	45.9
MG(2L,2L,4)	1485 (10)	33.4	2876 (9)	64.5
MG(1L,3L,4)	1440 (9)	28.5	3165 (10)	62.4
MG(5P,5P,4)	no convergence			
MG(1L,2L,4)	1494 (10)	25.0	3261 (10)	54.4
MG(1L,2L,5)	1511 (10)	25.0	3292 (11)	54.5
MG(1L,2L,6)	1510 (10)	25.0	3311 (11)	54.8
MG(1L,0L,5)	1691 (11)	15.5	3827 (12)	35.0
MG(1L,0L,6)	1704 (11)	15.5	3883 (12)	35.2

TABLE 2
MOSFET1 results: iterations and time (in minutes) for 20 Gummel iterations.

as the original finite difference matrix.[4] As the tables illustrate, the multigrid code with point relaxation is ineffective for the harder problems (though it works quite well on the JFET problem). On the other hand, the multigrid with line relaxation can be a very effective solver as it requires far less iterations than the corresponding ILU preconditioned code and that the overall run time is much better than the ILU code even though the cost per iteration is greater. Specifically, for the larger grid JFET problem and the smaller grid MOSFET1 and MOSFET2 problem the multigrid code is between a factor of 2 and 4 times faster than

[4] The original DANCIR code allows for ILU(1), ILU(2), etc. Over a wide variety of numerical experiments, we have found that the total run time is usually about the same using ILU(1) as opposed to ILU(0) (though the number of iterations is less using ILU(1)).

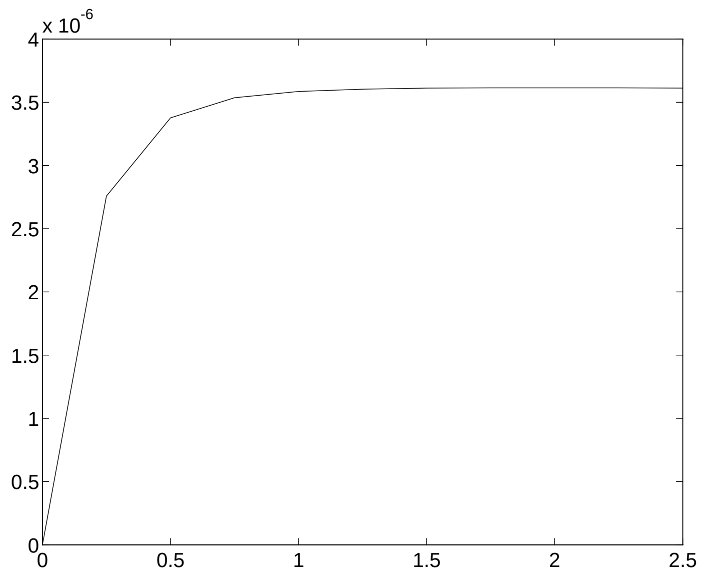


FIG. 7. *Current-voltage characteristics for MOSFET1.*

Preconditioner	321 × 97			
	Poisson		Continuity	
	its	time	its	time
ILU	21339 (61)	97.0	171015 (243)	770.2
MG(1L,0L,4)	3202 (9)	69.3	13774 (20)	287.9
MG(1L,1L,4)	2643 (8)	71.7	11212 (16)	295.5
MG(2L,2L,4)	2231 (6)	95.1	9336 (13)	389.3
MG(1L,3L,4)	2204 (6)	84.2	10569 (15)	392.6
MG(5P,5P,4)	no convergence			
MG(1L,2L,4)	2446 (7)	79.0	10352 (15)	325.6
MG(1L,2L,5)	2446 (7)	74.0	11961 (17)	359.0
MG(1L,2L,6)	2446 (7)	74.0	11024 (16)	331.8
MG(1L,0L,5)	3208 (9)	59.8	15593 (22)	289.0
MG(1L,0L,6)	3208 (9)	59.8	14390 (20)	267.0

TABLE 3
MOSFET2 results: iterations and time (in minutes) for 352 Gummel iterations.

the ILU code. More importantly, the number of iterations per linear solve is relatively independent of the grid size when using the multigrid preconditioner while it grows for the ILU scheme.[5] Thus, the savings associated with multigrid are even greater for larger grids.

In terms of storage, our multigrid scheme requires slightly more storage than the ILU method. Specifically, the ILU scheme requires the storage of the ILU preconditioner ($3n$ for 5-point symmetric difference operators where n is the number of grid points). In the multigrid scheme, we need additional storage for

[5] The number of multigrid iterations grows slightly for the JFET problem as the smaller grid is very easily solved by both the ILU and the multigrid schemes.

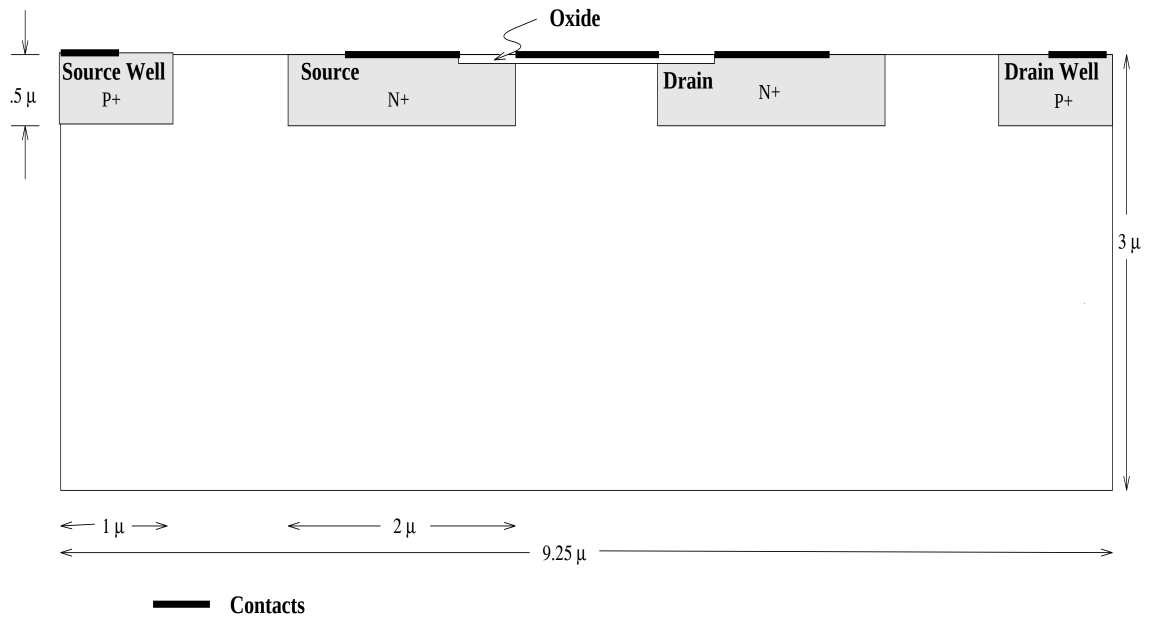


FIG. 8. *MOSFET2 Device*

	JFET		MOSFET1	MOSFET2
Preconditioner	49×33	321×97	129×129	321×97
ILU	5.1	321.7	231.5	945.8
MG(1L,0L,4)	4.3	114.4	75.1	424.0
MG(1L,0L,5)	4.3	99.5	68.3	406.5
MG(1L,0L,6)	4.3	100.1	68.5	391.1
MG(1L,1L,4)	4.8	142.2	85.4	432.5
MG(2L,2L,4)	6.1	153.1	118.0	555.2
MG(1L,3L,4)	5.2	139.1	110.6	544.2
MG(1L,2L,4)	5.3	126.8	98.6	470.9
MG(1L,2L,5)	5.3	122.7	98.6	499.7
MG(1L,2L,6)	5.6	124.1	99.1	472.5
MG(3P,0P,4)	5.9	92.5	nc	nc
MG(5P,5P,4)	7.1	126.3	nc	nc

TABLE 4
Total Simulation Time

the coarse grid versions of the matrix, the solution, and the right hand side ($\approx 3n/4$ for the coarse matrices and $\approx 2n/4$ for the coarse solutions and right hand sides). Thus, the ILU requires an additional $3n$ values while multigrid requires an additional $5n/4$ values. However, in our multigrid implementation we also store $2(4n/3)$ intermediate values for interpolation and projection operators and $2(4n/3)$ intermediate values for the factorization of the line solver.

We conclude this section by illustrating the total cpu time for each of device simulation in Table 4. By comparison with the earlier tables, the reader can verify that the conjugate gradient iteration dominates the calculation and thus the multigrid savings are significant with respect to the entire simulation time.

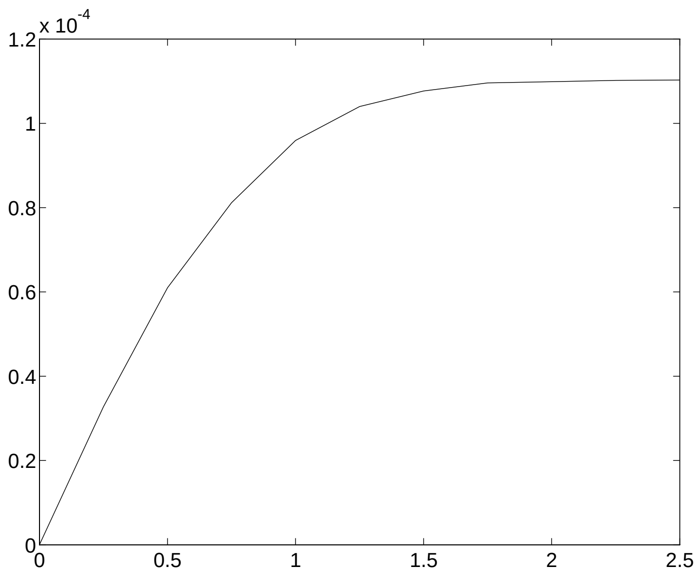


FIG. 9. *Current-voltage characteristics for MOSFET2.*

6. Conclusions. We have incorporated a multigrid preconditioner into a semiconductor device modeling code. To do this, a multigrid scheme was developed that could be used in cases that exhibited highly variable PDE coefficients and anisotropic behavior. In addition, special consideration had to be taken with respect to certain device characterisitics such as oxide layers and small contact regions. The resulting scheme is fast, parallel, and requires many fewer iterations than the ILU scheme that it replaced. On our sample problems we improved the performance by a factor of between 2 and 4 over the ILU preconditioner. Extensions to the 3-dimensional case are straight-forward and planned for the future.

Finally, we note that while we have used a multigrid solver for the linear equations that arise within Gummel's method, there are potentially much greater savings if the Gummel technique can be replaced by a nonlinear multigrid iteration. We have not pursued this, but we hope that this study will give insight into this possibility.

REFERENCES

[1] R. Alcouffe, A. Brandt, J. Dendy, and J. Painter. The multigrid method for diffusion equations with strongly discontinuous coefficients. *SIAM J. Sci. Stat. Comput.*, 2:430–454, 1981.

[2] B. Bachmann. A multigrid solver for the semiconductor equations. Technical report, Institut fur Angewandte Mathematik der Universitat Zurich, Ramistr. 74, 8001 Zurich, Switzerland, 1993.

[3] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31:333–390, 1977.

[4] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, 1983.

[5] W. Hackbusch. *Multi-grid Methods and Applications.* Springer-Verlag, Berlin, 1985.

[6] H.K.Gummel. A self-consistent iterative scheme for one-dimensional steady state transistor calculations. *IEEE Transactions on Electron Devices*, ED-11:455–465, 1964.

[7] J.C.Meza and J.F.Grcar. DANCIR: A three-dimensional steady-state semiconductor device simulator. Technical Report Sand89-8266, Sandia National Laboratories, Livermore, CA, 1990.

[8] R. Kettler and J. A. Meijerink. A multigrid method and a combined multigrid-conjugate gradient method for elliptic problems with strongly discontinuous coefficients in general domains. Technical Report Shell Publ. 604, KSEPL, Rijswijk, The Netherlands, 1981.

[9] M.S. Mock. *Analysis of Mathematical Models of Semiconductor Devices.* Boole Press, Dublin, 1983.

[10] J. Molenaar. *Multigrid Methods for Semiconductor Device Simulation.* PhD thesis, Centrum voor Wiskunde en Informatica, Amsterdam, 1992.
[11] Siegfried Selberherr. *Analysis and Simulation of Semiconductor Devices.* Springer-Verlag, New York, 1984.
[12] S.J.Polak, C.Den Heijer, W.H.A.Schilders, and P.Markowich. Semiconductor device modelling from the numerical point of view. *International Journal for Numerical Methods in Engineering*, 24:763–838, 1987.
[13] S.M. Sze. *Physics of Semiconductor Devices.* Wiley, New York, 1981.
[14] P. Wesseling. *An Introduction to Multigrid Methods.* Wiley, West Sussex, 1992.